ST. JOSEPH'S UNIVERSITY

BANGALORE-560027

ESTD-1882

COMPUTER SCIENCE

PRACTICAL RECORD

(DESIGN AND ANALYSIS OF ALGORITHM)

UNDER THE GUIDANCE OF

1. Prof. Sandhya

2. Prof. Bojamma

3. Prof. Sarakutty

4. Prof. Mrinmoyee

SUBMITTED BY:

Student Name: Shashank

Roll No: 222CSC33

ST. JOSEPH'S UNIVERSITY

BANGALORE

LABORATORY CERTIFICATE

This is to certify that, Sri. Shashank has satisfactorily completed the course of laboratory assignments in Design and Analysis of Algorithms prescribed by St. Joseph's University for the First Semester Master's degree course in Computer Science for the year 2022-23.

Name: Shashank

Reg. No.: 222CSC33

Date of Exam: 18/11/2022

Teacher In-Charge

Head of the Department/PG Coordinator

INDEX

1. **Write a program to implement linear search for a set of given elements**

```
/*-----------------------------------------------------------------
 @Author: Shashank
 Program to perform linear search on a set of elements
 -----------------------------------------------------------------*/
package Final_printable_code;

import java.util.ArrayList;
import java.util.Scanner;

public class LinearSearch {

    public static void linearSearch(int element, int[] array){
        boolean flag=false;
        for(int i=0;i<array.length;i++){
            if(element==array[i]){
                flag=true;
                System.out.println("Element found in the index "+i);
                break;
            }
        }
        if(flag==false) System.out.println("element not found");

    }
    //method overload for multi-linear search
    public static void linearSearch(int element, int[] array, boolean multiple){
        if(multiple){
            ArrayList<Integer> newArr = new ArrayList<>();
            for(int i=0;i<array.length;i++){
                if(element==array[i]){
                    newArr.add(i);
                }
            }
            if(newArr.isEmpty())
                System.out.println("element not found");
            else
                System.out.println("element found in");
                System.out.println(newArr.toString());

        }
        else{
            linearSearch(element, array);
        }
    }


    public static void main(String[] args) {
```

```
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number of elements in the array");
        int n = sc.nextInt();
        int[] array = new int[n];
        System.out.println("Enter the elements of the array");
        for(int i =0;i<n;i++){
            array[i] = sc.nextInt();
        }
        System.out.println("Enter the element to be searched in the array");
        int element=sc.nextInt();

        linearSearch(element, array);

    }
}
```

Output:

```
Enter the number of elements in the array
5
Enter the elements of the array
2 1 7 9 3
Enter the element to be searched in the array
7
Element found in the index 2
PS C:\Shanki\College\MSc Labs>
```

2. Write a program to implement binary search using divide and conquer technique.

```
/*-----------------------------------------------------------------
 @Author: Shashank
 Program to search for a given element in the array
 using binary search which uses divide and conquer method
 -----------------------------------------------------------------*/

package Final_printable_code;

import java.util.Arrays;
import java.util.Scanner;

public class BinarySearch {
    private static int binarySearchInternal(int element, int[] array, int min,
int max) {
        int mid = (min + max) / 2;
        // when the problem is not small
        if (max >= min) {
            if (array[mid] == element) {
                return mid;
            }
            // when the element is in the 1st half
            if (array[mid] > element) {
                max = mid - 1;
                return binarySearchInternal(element, array, min, max);
            }
            // when the element is in the second half
            else {
                min = mid + 1;
                return binarySearchInternal(element, array, min, max);
            }
        }
        // when element is not found
        return -1;
    }

    public static void binarySearch(int element, int[] array) {
        int min = 0;
        int max = array.length - 1;
        // binary search only works on sorted arrays so..
        Arrays.sort(array);
        int pos = binarySearchInternal(element, array, min, max);
        if (pos == -1) {
            System.out.println("Element not found");
        } else {
            System.out.println("Element found in position " + pos);
        }
```

```
    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number of elements in the array");
        int n = sc.nextInt();
        int[] array = new int[n];
        System.out.println("Enter the elements of the array");
        for (int i = 0; i < n; i++) {
            array[i] = sc.nextInt();
        }
        System.out.println("Enter the element to be searched in the array");
        int element = sc.nextInt();
        binarySearch(element, array);
    }
}
```

```
Enter the number of elements in the array
5
Enter the elements of the array
1 9 7 3 2
Enter the element to be searched in the array
3
Element found in position 2
PS C:\Shanki\College\MSc Labs>
```

3. Write a program to implement bubble sort for a set of given elements.

```
/*-----------------------------------------------------------------
 @Author: Shashank
 Program to sort the given array in ascending order
 using bubble sort
 -----------------------------------------------------------------*/

package Final_printable_code;

import java.util.Arrays;
import java.util.Scanner;

public class BubbleSort {

    public static int[] bubbleSort(int[] unsorted) {

        System.out.println("Using bubble sort");
        for (int i = 0; i < unsorted.length; i++) {
            boolean swapped = false;
            for (int j = 0; j < unsorted.length - 1 - i; j++) {
                if (unsorted[j] > unsorted[j + 1]) {
                    int t = unsorted[j];
                    unsorted[j] = unsorted[j + 1];
                    unsorted[j + 1] = t;
                    swapped = true;
                }
            }
            // if the swap did not take place the array is already sorted
            // hence we can break out of the loop to not do unnecessary
computation
            if (!swapped) {
                break;
            }
        }
        return (unsorted);
    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        // input the array
        System.out.println("Enter the number of elements in the array");
        int n = sc.nextInt();
        int[] array = new int[n];
        System.out.println("Enter the elements of the array");
        for (int i = 0; i < n; i++) {
            array[i] = sc.nextInt();
```

```
        }
        // printing the sorted array
        System.out.println("sorted array is " +
Arrays.toString(bubbleSort(array)));
    }
}
```

Output:

```
Enter the number of elements in the array
6
Enter the elements of the array
9 5 8 2 3 1
Using bubble sort
sorted array is [1, 2, 3, 5, 8, 9]
PS C:\Shanki\College\MSc Labs>
```

4. Write a program using selection sort to arrange n numbers.

```
/*-----------------------------------------------------------------
 @Author: Shashank
 Program to perform selection sort on a given unsorted array
 -----------------------------------------------------------------*/

package Final_printable_code;

import java.util.Arrays;
import java.util.Scanner;

public class SelectionSort {

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        // input of array
        System.out.println("Enter the number of elements in the array");
        int n = sc.nextInt();
        int[] array = new int[n];
        System.out.println("Enter the elements of the array");
        for (int i = 0; i < n; i++) {
            array[i] = sc.nextInt();
        }

        // sorting and printing the array
        System.out.println("sorted array is " +
Arrays.toString(selectionSort(array)));
    }

    public static int[] selectionSort(int[] unsorted) {

        System.out.println("Using selection sort");
        for (int i = 0; i < unsorted.length; i++) {
            int min_index = i;
            for (int j = i + 1; j < unsorted.length; j++) {
                if (unsorted[min_index] > unsorted[j]) {
                    min_index = j;
                }
            }
            int t = unsorted[i];
            unsorted[i] = unsorted[min_index];
            unsorted[min_index] = t;
        }
        return (unsorted);
    }
}
```

Output:

```
Enter the number of elements in the array
6
Enter the elements of the array
9 8 3 4 6 1
Using selection sort
sorted array is [1, 3, 4, 6, 8, 9]
PS C:\Shanki\College\MSc Labs>
```

5. Write a program to perform insertion sort to reorder n numbers in ascending order.

```
/*-----------------------------------------------------------------
 @Author: Shashank
 Program to sort a given array in ascending order using
 insertion sort algorithm
 -----------------------------------------------------------------*/
package Final_printable_code;

import java.util.Arrays;
import java.util.Scanner;

public class InsertionSort {
    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        // input of the array
        System.out.println("Enter the number of elements in the array");
        int n = sc.nextInt();
        int[] array = new int[n];
        System.out.println("Enter the elements of the array");
        for (int i = 0; i < n; i++) {
            array[i] = sc.nextInt();
        }
        // function call and printing the array
        System.out.println("sorted array is " +
Arrays.toString(insertionSort(array)));
    }

    public static int[] insertionSort(int[] array) {
        System.out.println("Using insertion sort to sort the array .. ");
        // starting from 2nd element as we assume 1st is sorted
        for (int i = 1; i < array.length; i++) {
            int key = array[i];
            int j = i - 1;
            while (j >= 0 && key < array[j]) {
                array[j + 1] = array[j];
                j--;
            }
            array[j + 1] = key;
        }
        return (array);
    }
}
Output:
```

```
Enter the number of elements in the array
5
Enter the elements of the array
9 8 7 5 2
Using insertion sort to sort the array ..
sorted array is [2, 5, 7, 8, 9]
```

6. Write a program to perform quick sort using divide and conquer technique to arrange n numbers.

```
/*-------------------------------------------------------------
 @Author: Shashank
 Program to perform quick sort and arrange a given array in
 ascending order
 -------------------------------------------------------------*/
package Final_printable_code;

import java.util.Arrays;
import java.util.Scanner;

public class QuickSort {
    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number of elements in the array");
        int n = sc.nextInt();
        int[] array = new int[n];
        System.out.println("Enter the elements of the array");
        for (int i = 0; i < n; i++) {
            array[i] = sc.nextInt();
        }
        System.out.println("sorted array is " + Arrays.toString(quickSort2(array,
0, array.length - 1)));
    }

    public static int[] quickSort2(int[] unsorted, int low, int high) {
        if (low < high) {
            int p = partition2(unsorted, low, high);
            quickSort2(unsorted, low, p - 1);
            quickSort2(unsorted, p + 1, high);
        }
        return unsorted;
    }

    // outputs the position on which array is split
    private static int partition2(int[] unsorted, int low, int high) {
        int i = low, j = high;

        int index = unsorted[low];
        while (i < j) {
            while (unsorted[i] < index && i < high) {
                i++;
            }
            while (unsorted[j] > index && j >= low) {
                j--;
```

```
            }
            if (i < j) {

                int temp = unsorted[i];
                unsorted[i] = unsorted[j];
                unsorted[j] = temp;
            }
        }
        int temp2 = unsorted[j];
        unsorted[j] = index;
        index = temp2;
        return j;
    }
}
```

Output:

```
\MSc Labs_9d41fb38\bin' 'Final_printable_code.QuickSort'
Enter the number of elements in the array
6
Enter the elements of the array
9 7 3 1 5 10
sorted array is [1, 3, 5, 7, 9, 10]
PS C:\Shanki\College\MSc Labs>
```

7. Write a program to perform merge sort on n number

```
/*-----------------------------------------------------------------
 @Author: Shashank
 Program to perform merge sort on a given unsorted array
 -----------------------------------------------------------------*/
package Final_printable_code;

import java.util.Arrays;
import java.util.Scanner;

public class MergeSort {
    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number of elements in the array");
        int n = sc.nextInt();
        int[] array = new int[n];
        System.out.println("Enter the elements of the array");
        for (int i = 0; i < n; i++) {
            array[i] = sc.nextInt();
        }

        System.out.println("sorted array is " + Arrays.toString(mergeSort(array,
0, array.length - 1)));
    }

    public static int[] mergeSort(int[] unsorted, int beg, int end) {
        if (beg < end) {
            int mid = (beg + end) / 2;
            // Divide the entire array into 2 halves and call merge sort on them
            mergeSort(unsorted, beg, mid);
            mergeSort(unsorted, mid + 1, end);
            merge(unsorted, beg, end, mid);
        }
        return unsorted;
    }

    private static void merge(int[] unsorted, int beg, int end, int mid) {
        int n1 = mid - beg + 1;
        int n2 = end - mid;
        int i, j, k = beg;
        int[] leftArray = new int[n1];
        int[] rightArray = new int[n2];

        // Copying the half of the arrays to 2 new temporary arrays
        for (i = 0; i < n1; i++)
            leftArray[i] = unsorted[beg + i];
```

```
        for (j = 0; j < n2; j++)
            rightArray[j] = unsorted[mid + j + 1];

        i = 0;
        j = 0;

        // comparing the left and right array and adding in order
        while (i < n1 && j < n2) {
            // count++;//to keep track of each comparisions
            if (leftArray[i] <= rightArray[j]) {
                unsorted[k] = leftArray[i];
                i++;
            } else {
                unsorted[k] = rightArray[j];
                j++;
            }
            k++;
        }

        // happens after either left or the right array runs
        // out and the remaining of the 2 are added
        while (i < n1) {
            unsorted[k] = leftArray[i];
            i++;
            k++;
        }
        while (j < n2) {
            unsorted[k] = rightArray[j];
            j++;
            k++;
        }

    }
}
```

Output:

```
\MSc Labs_9d41fb38\bin' 'Final_printable_code.MergeSort'
Enter the number of elements in the array
7
Enter the elements of the array
9 8 3 1 2 10 -5
sorted array is [-5, 1, 2, 3, 8, 9, 10]
PS C:\Shanki\College\MSc Labs>
```

8. Write a program to implement push and pop operations in a stack.

```
/*-------------------------------------------------------------------
 @Author: Shashank
 Program to demonstrate and implement the stack data structure
 -------------------------------------------------------------------*/
package Final_printable_code;

import java.util.ArrayList;

public class StackDemo {
    public ArrayList<Integer> array = new ArrayList<>();
    // delete the element at top
    public int pop() throws Exception {
        if (array.isEmpty()) {
            System.out.println("Array empty");
            throw new Exception("Array empty");
        }
        int ele = array.get(array.size() - 1);
        array.remove(array.size() - 1);
        return ele;
    }
    // insert element at the top
    public void push(int ele) {
        array.add(ele);
    }
    public void display() {
        System.out.println("Stack elements:"+array.toString());
    }
    public static void main(String[] args) throws Exception {
        StackDemo newStack = new StackDemo();
        newStack.push(6);
        newStack.push(5);
        newStack.display();
        newStack.pop();
        newStack.pop();
        newStack.display();
    }
}
```

Output:

```
63b8fec38e5714acfa\redhat.java\jdt_ws\MSc Labs_9d41fb38\bin' 'Final_printable_code.StackDemo'
Stack Elements :[6, 5]
Stack Elements :[]
PS C:\Shanki\College\MSc Labs>
```

9. Write a program to implement insert and delete operations in a queue.

```
/*-------------------------------------------------------------------
 @Author: Shashank
 Program to demonstrate implementation of a queue
 -------------------------------------------------------------------*/
package Final_printable_code;

public class QueueDemo {
    int front = -1;
    int rear = -1;
    int[] array;

    QueueDemo(int size) {
        this.array = new int[size];
    }

    QueueDemo() {
        this.array = new int[10];
    }

    public int deQueue() {
        if (isQempty()) {
            System.out.println("Queue is empty and cant delete");
            return -1;
        }
        // creating a backup of the deleted element just in case
        int t = array[front];
        if (front == rear) {
            front = rear = -1;
            return t;
        }
        front++;
        return t;
    }

    public void enQueue(int ele) {

        if (isQfull()) {
            System.out.println("Queue is full");
            return;
        }
        if (front == -1) {
            front = 0;
        }
        rear++;
        // adding the element to therea
        array[rear] = ele;
```

```
    }

    public boolean isQempty() {
        if (front == -1)
            return true;
        return false;
    }

    public boolean isQfull() {
        if (front == 0 && rear == array.length - 1)
            return true;
        return false;
    }

    public int peek() {
        return array[front];
    }

    public void display() {
        int i;
        for (i = front; i != rear; i = (++i) % array.length) {
            System.out.print(array[i] + " ");
        }
        System.out.print(array[i]);
        System.out.println("");
        return;
    }

    public static void main(String[] args) {

        QueueDemo newQueue = new QueueDemo(5);

        newQueue.enQueue(5);
        newQueue.enQueue(9);
        newQueue.enQueue(4);
        newQueue.display();
        newQueue.deQueue();
        newQueue.display();
    }
}
```

Output:

```
\MSc Labs_9d41fb38\bin' 'Final_printable_code.QueueDemo'
5 9 4
9 4
PS C:\Shanki\College\MSc Labs>
```

10. Write a program to find maximum and minimum of an array using divide and conquer algorithm.

```
/*-----------------------------------------------------------------
 @Author: Shashank
 Program to find the maximum and the minimum of a given array using divide
 and conquor technique for a given matrix
 -----------------------------------------------------------------*/
package Final_printable_code;

import java.util.Scanner;

public class MaxMinDandC {

    public static int[] minMaxSearch(int[] array, int min, int max) {
        int i = min, j = max;
        // The array contains local minimum at position 0 and local maximum at
position
        // 1
        int[] minmax_array = new int[2];

        // if its a small problem of length 1
        if (i == j) {
            minmax_array[0] = array[i];
            minmax_array[1] = array[i];
            return minmax_array;
        }

        // if its a small problem of length 2
        if (i == j - 1) {
            if (array[i] < array[j]) {
                minmax_array[0] = array[i];
                minmax_array[1] = array[j];
            } else {
                minmax_array[1] = array[i];
                minmax_array[0] = array[j];
            }
            return minmax_array;
        }
        // if its not a small problem then use the divide and conquor method
        int mid = (min + max) / 2;
        int[] left_minmax = minMaxSearch(array, min, mid);
        int[] right_minmax = minMaxSearch(array, mid + 1, max);

        if (left_minmax[0] < right_minmax[0])
            minmax_array[0] = left_minmax[0];
        else
            minmax_array[0] = right_minmax[0];
```

```
        if (left_minmax[1] > right_minmax[1])
            minmax_array[1] = left_minmax[1];
        else
            minmax_array[1] = right_minmax[1];

        return minmax_array;
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        System.out.println("Enter the number of elements in the array");
        int n = sc.nextInt();
        int[] array = new int[n];
        System.out.println("Enter all the elements of the array");
        for (int i = 0; i < n; i++) {
            array[i] = sc.nextInt();
        }
        int[] mmarr = new int[2];
        mmarr = minMaxSearch(array, 0, array.length - 1);

        System.out.println("Minimum of the array is " + mmarr[0]);
        System.out.println("Maximum of the array is " + mmarr[1]);

    }
}
```

Output:

```
63b8fec38e5714acfa\redhat.java\jdt_ws\MSc Labs_9d41fb38\bin' 'Final_printable_code.MaxMinDand
Enter the number of elements in the array
5
Enter all the elements of the array
9 3 1 8 -4
Minimum of the array is -4
Maximum of the array is 9
PS C:\Shanki\College\MSc Labs>
```

11. Write a program to implement 0/1 Knapsack Problem using Dynamic Programming.

```
package Final_printable_code;

import java.util.*;

class NewKnapSack {
    int totpro, capacity;

    void knapsck(int w[], int ob[], int p[], int Case, float piwi[], int tempcap,
int x[], int n) {
        int i, k, temppro = 0, j;

        capacity = tempcap;

        if (Case <= 3) {
            for (i = 0; i < n; i++)
                x[i] = 0;

            sort(w, ob, p, n, Case, piwi);

            for (i = 0; i < n; i++) {
                if (tempcap >= w[i]) {
                    tempcap = tempcap - w[i];
                    k = ob[i];
                    x[k - 1] = 1;
                }
            }

            for (i = 0; i < n; i++)
                for (j = 0; j < n; j++)
                    if (i + 1 == ob[j])
                        temppro = temppro + (x[i] * p[j]);

            System.out.print("\nSolution Set For Case " + Case + ":X= ");
            for (i = 0; i < n; i++)
                System.out.print(" " + x[i]);

            System.out.println("\n Case " + Case + " Profit is " + temppro);

            Case++;

            if (totpro < temppro)
                totpro = temppro;

            knapsck(w, ob, p, Case, piwi, capacity, x, n);
        } else
            System.out.println("\n\n Optimal Profit is " + totpro);
```

```
}

void swap(int a[], int i, int j) {
    int temp;

    temp = a[i];
    a[i] = a[j];
    a[j] = temp;
}

void swap1(float a[], int i, int j) {
    float temp;

    temp = a[i];
    a[i] = a[j];
    a[j] = temp;
}

void sort(int w[], int ob[], int p[], int n, int cas, float piwi[]) {
    int i, j;

    for (i = 0; i < n; i++)
        for (j = i + 1; j < n; j++) {
            if (w[i] >= w[j] && cas == 1) {
                swap(w, i, j);

                swap(ob, i, j);

                swap(p, i, j);

                swap1(piwi, i, j);
            }
            if (p[i] <= p[j] && cas == 2) {
                swap(p, j, i);
                swap(ob, i, j);
                swap(w, i, j);
                swap1(piwi, i, j);
            }
            if (piwi[i] <= piwi[j] && cas == 3) {
                swap1(piwi, j, i);
                swap(p, i, j);
                swap(ob, i, j);
                swap(w, i, j);
            }
        }
}
public static void main(String args[]) {
    int n, m, w[], p[], i, ob[], x[];
```

```
        float piwi[], a, b;
        w = new int[50];
        p = new int[50];
        ob = new int[50];
        x = new int[50];
        piwi = new float[50];
        Scanner sc = new Scanner(System.in);
        System.out.println("\n Enter the Value of N: ");
        n = sc.nextInt();
        System.out.println("\n Enter the Capacity: ");
        m = sc.nextInt();
        System.out.println("\n Enter " + n + " Weigths: ");
        for (i = 0; i < n; i++) {
            w[i] = sc.nextInt();
            ob[i] = i + 1;
        }
        System.out.println("\n Enter " + n + " Profits: ");
        for (i = 0; i < n; i++)
            p[i] = sc.nextInt();
        for (i = 0; i < n; i++) {
            a = p[i];
            b = w[i];
            piwi[i] = a / b;
        }
        NewKnapSack ks = new NewKnapSack();
        ks.knapsck(w, ob, p, 1, piwi, m, x, n);
    }
}
```

Output:

```
 Enter the Value of N:
7

 Enter the Capacity:
15

 Enter 7 Weigths:
4 2 3 2 4 2 1

 Enter 7 Profits:
2 3 15 10 5 8 4

Solution Set For Case 1:X=  0 1 1 1 1 1 1
 Case 1 Profit is 45

Solution Set For Case 2:X=  0 1 1 1 1 1 1
 Case 2 Profit is 45

Solution Set For Case 3:X=  0 1 1 1 1 1 1
 Case 3 Profit is 45


 Optimal Profit is 45
```

12. Write a program to display the degree of vertices for an edge matrix for a particular graph G.

```
/*-------------------------------------------------------------
 @Author: Shashank
 Program to find the degree of a node for a given graph
 -------------------------------------------------------------*/
package Final_printable_code;

/*Works on both directed and non directed graphs */

import java.util.Scanner;

public class NodeDegree {

    // private method to input edge_matrix and output the degree array of nodes
    private static int[] getDegreeArray(int[][] edge_matrix) {

        int n = edge_matrix.length;
        int[] degree_array = new int[n];
        for (int i = 0; i < n; i++) {
            int local_count = 0;
            for (int j = 0; j < n; j++) {
                if (edge_matrix[i][j] != 999 && edge_matrix[i][j] >= 0) {
                    local_count++;
                }
            }
            degree_array[i] = local_count;
        }
        return degree_array;
    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);

        System.out.println("Enter the number of vertices");
        int n = sc.nextInt();

        int[][] edge_matrix = new int[n][n];

        System.out.println("Enter the edge adjacency matrix");
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                edge_matrix[i][j] = sc.nextInt();
                // idiot proofing to allow 0 as input in vertex to itself
```

```
                if (i == j) {
                    edge_matrix[i][j] = 999;
                }
            }
        }

        int[] cost_array = getDegreeArray(edge_matrix);
        for (int i = 0; i < n; i++) {
            System.out.println("the degree of node " + i + " is " +
cost_array[i]);
        }
    }
}
```

Output:

```
Enter the number of vertices
7
Enter the edge adjacency matrix
999 1 999 999 999 999 999
1 999 1 1 999 999 999
999 1 999 999 1 999 999
1 1 999 999 999 999 999
999 999 1 999 999 999 999
999 999 999 999 999 999 1
999 999 999 999 999 1 999
the degree of node 0 is 1
the degree of node 1 is 3
the degree of node 2 is 2
the degree of node 3 is 2
the degree of node 4 is 1
the degree of node 5 is 1
the degree of node 6 is 1
PS C:\Shanki\College\MSc Labs>
```

13. Write a program to find the shortest path for a given vertex in a weighted connected graph using Dijkstra's algorithm.

```
/*-----------------------------------------------------------------
 @Author: Shashank
 Program to find the minimum cost to visit all the nodes from
 a single source using dijikstras algorithm
 -----------------------------------------------------------------*/

package Final_printable_code;

import java.util.Arrays;
import java.util.Scanner;

public class DijikstrasAlgo {

  public static void dijkstra(int[][] graph, int source) {
    int count = graph.length;
    boolean[] visitedVertex = new boolean[count];
    int[] distance = new int[count];

    Arrays.fill(visitedVertex, false);
    Arrays.fill(distance, 999);

    // Distance of self loop is zero
    distance[source] = 0;
    for (int i = 0; i < count; i++) {

      // Update the distance between neighbouring vertex and source vertex
      int u = findMinDistance(distance, visitedVertex);
      visitedVertex[u] = true;

      // Update all the neighbouring vertex distances
      for (int v = 0; v < count; v++) {
        if (!visitedVertex[v] && graph[u][v] != 0 && (distance[u] + graph[u][v] <
distance[v])) {
          distance[v] = distance[u] + graph[u][v];
        }
      }
    }
    for (int i = 0; i < distance.length; i++) {
      System.out.println(String.format("Distance from %s to %s is %s", source, i,
distance[i]));
    }

  }

  // Finding the minimum distance
```

```
  private static int findMinDistance(int[] distance, boolean[] visitedVertex) {
    int minDistance = 999;
    int minDistanceVertex = -1;
    for (int i = 0; i < distance.length; i++) {
      if (!visitedVertex[i] && distance[i] < minDistance) {
        minDistance = distance[i];
        minDistanceVertex = i;
      }
    }
    return minDistanceVertex;
  }

  public static void main(String[] args) {

    Scanner sc = new Scanner(System.in);

    System.out.println("Enter the number of vertices");
    int n = sc.nextInt();

    int[][] cost_matrix = new int[n][n];

    System.out.println("Enter the cost adjacency matrix");
    for (int i = 0; i < n; i++) {
      for (int j = 0; j < n; j++) {
        cost_matrix[i][j] = sc.nextInt();
        // idiot proofing to allow 0 as input in vertex to itself
        if (i == j) {
          cost_matrix[i][j] = 999;
        }
      }
    }
    DijikstrasAlgo T = new DijikstrasAlgo();
    T.dijkstra(cost_matrix, 0);
  }
}
```

Output:

```
Enter the number of vertices
5
Enter the cost adjacency matrix
999 11 9 7 8
11 999 15 14 13
9 15 999 12 14
7 14 12 999 6
8 13 14 6 999
Distance from 0 to 0 is 0
Distance from 0 to 1 is 11
Distance from 0 to 2 is 9
Distance from 0 to 3 is 7
Distance from 0 to 4 is 8
PS C:\Shanki\College\MSc Labs>
```

14. Write a program to find the minimum cost spanning tree of a given undirected graph using Kruskal's algorithm

```
/*-----------------------------------------------------------------
 @Author: Shashank
 Program to find the minimum spanning tree of a given graph
 using the kruskal's algorithm
 -----------------------------------------------------------------*/
package Final_printable_code;

import java.util.*;

public class Kruskals {
    static class Edge {
        int source;
        int destination;
        int weight;

        public Edge(int source, int destination, int weight) {
            this.source = source;
            this.destination = destination;
            this.weight = weight;
        }
    }

    static class Graph {
        int vertices;
        ArrayList<Edge> allEdges = new ArrayList<>();

        Graph(int vertices) {
            this.vertices = vertices;
        }

        public void addEgde(int source, int destination, int weight) {
            Edge edge = new Edge(source, destination, weight);
            allEdges.add(edge);
        }

        public void kruskalMST() {
            PriorityQueue<Edge> pq = new PriorityQueue<>(allEdges.size(),
                    Comparator.comparingInt(o -> o.weight));
            // adding all edges to a priority queue which sorts according to the
edge
            // weights
            pq.addAll(allEdges);
            int[] parent = new int[vertices];
            // makes an array of all vertices
            makeSet(parent);
```

```
            ArrayList<Edge> mst = new ArrayList<>();
            int index = 0;
            // start adding edges to the mst according to lowest weights
            while (index < vertices - 1) {
                Edge edge = pq.remove();
                int x_set = find(parent, edge.source);
                int y_set = find(parent, edge.destination);
                // if forms a loop do nothing
                if (x_set == y_set) {
                }
                // else add them to mst
                else {
                    mst.add(edge);
                    index++;
                    union(parent, x_set, y_set);
                }
            }
            System.out.println("Minimum Spanning Tree: ");
            printGraph(mst);
        }

        public void makeSet(int[] parent) {
            for (int i = 0; i < vertices; i++) {
                parent[i] = i;
            }
        }

        public int find(int[] parent, int vertex) {
            if (parent[vertex] != vertex)
                return find(parent, parent[vertex]);
            return vertex;
        }

        public void union(int[] parent, int x, int y) {
            int x_set_parent = find(parent, x);
            int y_set_parent = find(parent, y);
            parent[y_set_parent] = x_set_parent;
        }

        public void printGraph(ArrayList<Edge> edgeList) {
            for (int i = 0; i < edgeList.size(); i++) {
                Edge edge = edgeList.get(i);
                System.out.println("Edge-" + i + " source: " + edge.source + "
destination: " + edge.destination
                        + " weight: " + edge.weight);
            }
        }
    }
```

```
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        // Input the number vertices and edges
        System.out.println("Enter the number of vertices initially");
        int n = sc.nextInt();
        Graph graph = new Graph(n);

        System.out.println("Enter the number of edges present");
        int edges = sc.nextInt();
        // input the edges
        for (int i = 0; i < edges; i++) {
            System.out.println("Enter the source, destination and weight of edge
" + (i + 1));
            int s = sc.nextInt();
            int d = sc.nextInt();
            int w = sc.nextInt();
            graph.addEgde(s, d, w);
        }

        graph.kruskalMST();
    }
}
```

Output:

```
Enter the number of vertices initially
5
Enter the number of edges present
4
Enter the source, destination and weight of edge 1
0 1 10
Enter the source, destination and weight of edge 2
0 2 8
Enter the source, destination and weight of edge 3
1 3 7
Enter the source, destination and weight of edge 4
1 4 10
Minimum Spanning Tree:
Edge-0 source: 1 destination: 3 weight: 7
Edge-1 source: 0 destination: 2 weight: 8
Edge-2 source: 1 destination: 4 weight: 10
Edge-3 source: 0 destination: 1 weight: 10
PS C:\Shanki\College\MSc Labs>
```

15. Write a program to print all reachable nodes given a starting node in a digraph using BFS algorithm.

```
/*-----------------------------------------------------------------
 @Author: Shashank
 Program to find all reachable nodes from a source node using BFS
 -----------------------------------------------------------------*/
package Final_printable_code;

import java.util.ArrayList;
import java.util.Scanner;

public class TraversalBFS {
    int n;
    boolean[] visited;

    TraversalBFS(int n) {
        visited = new boolean[n];
    }

    ArrayList<Integer> traversal_arr = new ArrayList<>();

    int front = 0;

    void rBFS(int[][] cost_matrix, int start_node) {
        n = cost_matrix.length;
        this.visited[start_node] = true;
        if (!traversal_arr.contains(start_node))
            traversal_arr.add(start_node);

        for (int i = 0; i < n; i++) {
            if (cost_matrix[start_node][i] != 999 && !visited[i]) {
                // add all unvisted neighbours to the traversal array
                if (!traversal_arr.contains(i))
                    traversal_arr.add(i);
            }
        }
        // if there are elements in the traversal array who have not been
searched
        // through yet then ..
        if (traversal_arr.size() > front + 1)
            rBFS(cost_matrix, traversal_arr.get(++front));

    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number of vertices");
        int n = sc.nextInt();
```

```
        int[][] cost_matrix = new int[n][n];

        System.out.println("Enter the edge matrix");
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                cost_matrix[i][j] = sc.nextInt();
                if (i == j) {
                    cost_matrix[i][j] = 999;
                }
            }
        }

        System.out.println("Enter starting node");
        int s_node = sc.nextInt();

        TraversalBFS gs1 = new TraversalBFS(cost_matrix.length);
        gs1.rBFS(cost_matrix, s_node);
        System.out.println("Traversal order in breadth first search is " +
(gs1.traversal_arr));
    }
}
```

Output:

```
Enter the number of vertices
7
Enter the edge matrix
999 1 999 999 999 999 999
1 999 1 1 999 999 999
999 1 999 999 1 999 999
1 1 999 999 999 999 999
999 999 1 999 999 999 999
999 999 999 999 999 999 1
999 999 999 999 999 1 999
Enter starting node
0
Traversal order in breadth first search is [0, 1, 2, 3, 4]
PS C:\Shanki\College\MSc Labs>
```

16. Write a program to check whether the given graph is connected or not using DFS algorithm.

```
/*-----------------------------------------------------------------
 @Author: Shashank
 Program to find the degree of a node for a given graph
 -----------------------------------------------------------------*/
package Final_printable_code;

import java.util.ArrayList;
import java.util.Scanner;

public class TraversalDFS {

    int n;
    boolean[] visited;

    TraversalDFS(int n) {
        visited = new boolean[n];
    }

    ArrayList<Integer> traversal_arr = new ArrayList<>();

    int front = 0;

    void DFS(int[][] cost_matrix, int start_node) {
        n = cost_matrix.length;
        this.visited[start_node] = true;
        traversal_arr.add(start_node);
        if (isUnvisted(visited)) {
            for (int i = 0; i < n; i++) {
                if (cost_matrix[start_node][i] != 999 && !visited[i]) {
                    // if the neighbour is not visited then call dfs on it
                    DFS(cost_matrix, i);
                }
            }
        }
    }

    // checks if there are unvisited vertices remaining
    private static boolean isUnvisted(boolean[] arr) {
        for (boolean x : arr) {
            if (x == false) {
                return true;
            }
        }
        return false;
    }
```

```
    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number of vertices");
        int n = sc.nextInt();

        int[][] cost_matrix = new int[n][n];

        System.out.println("Enter the edge matrix");
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                cost_matrix[i][j] = sc.nextInt();
                if (i == j) {
                    cost_matrix[i][j] = 999;
                }
            }
        }

        System.out.println("Enter starting node");
        int s_node = sc.nextInt();

        TraversalDFS gs1 = new TraversalDFS(cost_matrix.length);
        gs1.DFS(cost_matrix, s_node);
        System.out.println("Traversal order in breadth first search is " +
(gs1.traversal_arr));
        // The visited array contains booleans for each vertices
        if (isUnvisted(gs1.visited)) {
            System.out.println("The graph is disjoint");
        } else
            System.out.println("All nodes are reachable");

    }
}
```

Output:

```
Enter the number of vertices
7
Enter the edge matrix
999 1 999 999 999 999 999
1 999 1 1 999 999 999
999 1 999 999 1 999 999
1 1 999 999 999 999 999
999 999 1 999 999 999 999
999 999 999 999 999 999 1
999 999 999 999 999 1 999
Enter starting node
0
Traversal order in breadth first search is [0, 1, 2, 4, 3]
The graph is disjoint
```

17. Write a program to compute the transitive closure of a given directed graph using Warshall's algorithm.

```
/*-----------------------------------------------------------------
 @Author: Shashank
 Program to find the transitive closure edge matrix
 for a given matrix
 -----------------------------------------------------------------*/

package Final_printable_code;

import java.util.ArrayList;
import java.util.List;
import java.util.Scanner;

//Edge class describes an edge with source and destination properties
class Edge {
    int source, dest;

    public Edge(int source, int dest) {
        this.source = source;
        this.dest = dest;
    }
}

class Graph {
    // contains a matrix which represents the graph
    List<List<Integer>> adjList = null;

    // constructor to initialize the graph
    Graph(List<Edge> edges, int N) {
        adjList = new ArrayList<>(N);

        for (int i = 0; i < N; i++) {
            adjList.add(i, new ArrayList<>());
        }
        for (int i = 0; i < edges.size(); i++) {
            int src = edges.get(i).source;
            int dest = edges.get(i).dest;

            adjList.get(src).add(dest);
        }
    }
}

public class TransClosure {
    public static void DFS(Graph graph, byte[][] C, int root, int descendant) {
        for (int child : graph.adjList.get(descendant)) {
```

```
            if (C[root][child] == 0) {

                C[root][child] = 1;
                DFS(graph, C, root, child);
            }
        }
    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
        System.out.println("Enter the number of edges initially");
        int n = sc.nextInt();

        // taking all the edges present as input
        List<Edge> edges = new ArrayList<>();
        for (int i = 0; i < n; i++) {
            System.out.println("Edge " + (i + 1));
            System.out.println("Enter source");
            int source = sc.nextInt();
            System.out.println("Enter destination");
            int destination = sc.nextInt();
            edges.add(new Edge(source, destination));
        }

        final int N = n + 1;
        // initalizing the graph
        Graph graph = new Graph(edges, N);
        // creating the byte array representing the output graph
        byte[][] C = new byte[N][N];
        System.out.println("TRANSITIVE CLOSURE:-\n");
        for (int v = 0; v < N; v++) {
            C[v][v] = 1;
            // using depth first search on each vertex to visit all children
possible
            DFS(graph, C, v, v);
            // printing each line
            for (int u = 0; u < N; u++)
                System.out.print(C[v][u] + " ");
            System.out.println();
        }
    }
}
```

Output:

```
Enter the number of edges initially
4
Edge 1
Enter source
0
Enter destination
1
Edge 2
Enter source
0
Enter destination
2
Edge 3
Enter source
2
Enter destination
3
Edge 4
Enter source
2
Enter destination
4
TRANSITIVE CLOSURE:-

1 1 1 1 1
0 1 0 0 0
0 0 1 1 1
0 0 0 1 0
0 0 0 0 1
```

18. **Write a program to implement all pair shortest path's problem using Floyd's Algorithm**.

```
/*-----------------------------------------------------------------
 @Author: Shashank
 Program to find the all pair shortest path using the
 Floyd-Warshall's algorithm (dynamic programming)
 -----------------------------------------------------------------*/
package Final_printable_code;

import java.util.Scanner;

class FloydWarshall {

    void floydWarshall(int graph[][], int n) {
        int dist[][] = new int[n][n];
        int i, j, k;
        for (i = 0; i < n; i++)
            for (j = 0; j < n; j++)
                dist[i][j] = graph[i][j];
        // k counts the cycles
        for (k = 0; k < n; k++) {
            // traversing the graph with i and j
            for (i = 0; i < n; i++) {
                for (j = 0; j < n; j++) {
                    if (dist[i][k] + dist[k][j] < dist[i][j])
                        dist[i][j] = dist[i][k] + dist[k][j];
                }
            }
        }
        printSolution(dist, n);
    }

    void printSolution(int dist[][], int n) {
        System.out.println("SHORTEST PATH MATRIX:-");
        for (int i = 0; i < n; ++i) {
            for (int j = 0; j < n; ++j) {
                if (dist[i][j] == 999)
                    System.out.print("I ");
                else
                    System.out.print(dist[i][j] + " ");
            }
            System.out.println();
        }
    }

    public static void main(String[] args) {

        Scanner sc = new Scanner(System.in);
```

```
        System.out.println("Enter the number of vertices");
        int n = sc.nextInt();

        int[][] cost_matrix = new int[n][n];

        // taking cost matrix of graph as input
        System.out.println("Enter the cost matrix");
        for (int i = 0; i < n; i++) {
            for (int j = 0; j < n; j++) {
                cost_matrix[i][j] = sc.nextInt();
                if (i == j) {
                    cost_matrix[i][j] = 0;
                }
            }
        }

        FloydWarshall a = new FloydWarshall();
        a.floydWarshall(cost_matrix, n);
    }
}
```

Output:

```
Enter the number of vertices
3
Enter the cost matrix
0 4 11
6 0 2
3 999 0
SHORTEST PATH MATRIX:-
0 4 6
5 0 2
3 7 0
PS C:\Shanki\College\MSc Labs>
```